Theoretical Computer Science 404 (2008) 202–218

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Embedding finite automata within regular expressions☆

Shoham Ben-David [a], Dana Fisman [b,c,*], Sitvanit Ruah [c]

[a] University of Waterloo, Canada
[b] Weizmann Institute of Science, Israel
[c] IBM Haifa Research Lab, Israel

ARTICLE INFO

Keywords:
Regular expression
Finite automata
Model checking
Linear translation
On-the-fly
Embedding
PSL

ABSTRACT

Regular expressions and their extensions have become a major component of industry-oriented specification languages such as IEEE PSL [IEEE Standard for Property Specification Language (PSL). IEEE Std 1850™-2005]. The model checking procedure of regular expression based formulas, involves constructing an automaton which runs in parallel with the model.

In this paper we re-examine the automata construction. We propose an algorithm that allows an intermediate representation mixing both regular expressions and automata. This representation can be thought of as plugging an automaton inside a regular expression, to replace an existing sub-expression. In order to be verified, the intermediate representation is then translated into another automaton, resulting in a set of automata running in parallel. A key feature of this algorithm is that the plug-in automaton is independent of the regular expression from which it originated, and thus can be used in several different properties.

We demonstrate the usefulness of our method by providing a set of applications. We show how the use of our method allows *modularity* and *flexibility* of the automata construction, and can increase *expressiveness* when SEREs are mixed with CTL. We give two applications for which it significantly reduces the *size* of the automata built for formulas, thus reducing the overall run time of the model checking procedure.

1. Introduction

Symbolic model checking has been found extremely efficient in the verification of hardware designs, and has been widely adopted in industry in recent years. While traditional model checkers [15] used the temporal logics CTL [9] or LTL [17] as their specification language, contemporary industrial languages, have sought ways to make the specification language easier to learn and use. The IEEE standard language PSL [11] augments LTL with the use of Extended Regular Expressions (SEREs, using the formulation of [11]). SEREs extend traditional regular expressions (REs) with several operators (see section 2.2) which do not increase the expressive power, but increase succinctness.

SEREs are a convenient means to describe a sequence of Boolean events which may constitute a behavior of the model. For example, the formula $\varphi = \{req \cdot \neg ack^* \cdot ack\}!$ asserts that on all execution paths of the model, *req* is active on the first cycle, *ack* is then inactive for zero or more cycles, and then *ack* becomes active. Similarly, regular expressions can be used to describe an *undesirable* behavior of the model. The formula not $\{req \cdot \neg ack^* \cdot ack\}!$ asserts that there *does not exist* an execution path on which *req* is active on the first cycle, *ack* is then inactive for zero or more cycles, and then *ack* becomes active.

☆ This work was supported in part by the European Commission (FP6 STREP PROSYD contract no. 507219).
* Corresponding author at: Weizmann Institute of Science, Israel.
E-mail address: danafi@cs.huji.ac.il (D. Fisman).

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.03.011

In this paper we consider formulas given as not SERE!. These formulas are of special interest for several reasons. First, it is a convenient way for specification which is widely used. Second, a large subset of properties in LTL, CTL and their SERE extensions RLTL and RCTL can be automatically reduced to an equivalent formula of the form not r! [4,5,14]. Finally, these properties enjoy an efficient model checking method, as explained below. A not r! formula has the nature that it is sufficient to find one execution path of the model satisfying r, in order to conclude that the formula does not hold in the model. This nature allows a not r! formula to be modeled by a non-deterministic finite automaton (NFA) N_r, which accepts sequences satisfying r, and which is *linear* in the size of r. Running it in parallel with the model, we then verify the invariant property AG $\neg bad$ where *bad* is a Boolean expression asserting that N_r is in an accepting state. The reduction to an invariant property is important, since invariant properties are easier to verify. The efficient verification algorithm, together with the wide variety of properties which can be transformed into not SERE! formulas, has made them the major translation path of the IBM model checking tool-set RuleBase [3].

The translation of a not SERE! formula into an automaton is re-examined in this paper. Traditionally, a not SERE! formula is directly translated into an automaton. We propose an embedding algorithm that allows the translation to be modular, by introducing an intermediate representation of the formula, mixing both SEREs and automata. Let r be a SERE and t a sub-SERE of r. We show a process in which t is plugged out of r, a separate side-automaton is built for t, and a simpler SERE referring to the side-automaton is plugged into r, replacing t. A key feature of this algorithm is the fact that the side-automaton built is *independent* of the SERE it originated from. This allows the side-automaton to be referred to by a SERE that can be plugged into any other SERE when appropriate.

We emphasize that the embedding algorithm is not necessarily better than traditional methods for every formula. However, it is very useful in many cases, and we demonstrate this by a set of applications. We show how the use of this method allows *modularity* and *flexibility* of the automata construction, and can increase expressiveness when SEREs are mixed with CTL. We give two applications for which it significantly reduces the *size* of the automata built for formulas, thus reducing the overall run time of the model checking procedure.

The rest of the paper is organized as follows. Section 2 covers some preliminaries. Section 3 gives our embedding algorithm, and in Section 4 we discuss the benefits of the algorithm and give example applications. Section 5 concludes the paper. The proofs appear in Appendix.

2. Preliminaries

2.1. The computational model — DTS

We represent a finite state program by a *discrete transition system* (DTS). A DTS can represent both a Kripke structure (with no fairness constraints) and a finite automaton on finite words (NFA). We have chosen to work with a DTS for several reasons: (1) they allow a uniform way to compose in parallel a Kripke structure representing a system with NFAs as we construct for formulas. (2) The symbolic nature of the DTS allows us to refer to states of one automata in the transitions/states of another automata, a property which is needed for our construction.

The definition of a DTS is derived from the definition of a fair discrete system (FDS) [12].

Definition 1. A DTS $\mathcal{D} : \langle V, \Theta, \rho, \mathcal{A} \rangle$ consists of the following components:

- $V = \{u_1, \ldots, u_n\}$: A finite set of typed *state variables* over possibly infinite domains. We define a *state* s to be a type-consistent interpretation of V, assigning to each variable $u \in V$ a value $s[u]$ in its domain. We denote by Σ_V the set of all states, and by B_V the set of all Boolean expressions over the state variables in V (as Boolean operators we allow, on top of the ordinary "and", "or" and "not", equality between variables of the same type). When V is understood from the context we write simply Σ and B, respectively.

Example 2. Let V denote the set $\{a, b, c\}$ of Boolean state variables. Then Σ_V the set of interpretation of these variables contains the 8 interpretations giving different truth values to a, b and c. That is, $\Sigma_V = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$. B_V is the set of all Boolean expressions over these variables. For example, the boolean expressions a, $a \vee b$, $\neg c$ and $(a \rightarrow \neg b) \wedge c$ are all in B_V.

- Θ: The *initial condition*. This is an assertion characterizing all the initial states of the DTS.
- ρ: The *transition relation*. This is an assertion $\rho(V, V')$ relating a state $s \in \Sigma_V$ to its $\mathcal{D}$-successor $s' \in \Sigma_V$ by referring to both unprimed and primed versions of the state variables. The transition relation $\rho(V, V')$ identifies state s' as a $\mathcal{D}$-successor of state s if $\langle s, s' \rangle \models \rho(V, V')$, where $\langle s, s' \rangle$ is the joint interpretation which interprets $u \in V$ as $s[u]$ and u' as $s'[u]$.
- $\mathcal{A}$: The *accepting condition* for finite words. This is an assertion characterizing all the accepting states for runs of the DTS satisfying finite words.

Let $\mathcal{D}$ be a DTS for which the above components have been identified. We define a *run* of $\mathcal{D}$ to be a finite or infinite non-empty sequence of states $\sigma : s_0 s_1 s_2 \ldots$ satisfying the requirements of *initiality* i.e. that $s_0 \models \Theta$; and of *consecution* i.e. that for each $j = 0, 1, \ldots$, the state s_{j+1} is a $\mathcal{D}$-successor of state s_j. A run satisfying the requirement of *maximality* i.e. that it is either

infinite, or terminates at a state s_k which has no $\mathcal{D}$-successors is termed a *maximal run*. Let $U \subseteq V$ be a subset of the state variables. A run $\sigma : s_0 s_1 s_2 \ldots s_n \ldots$ is said to be *satisfying a finite word* $w = b_0 b_1 \ldots b_n$ over B_U iff for every i, $0 \leq i \leq n$, $s_i \models b_i$. A run $\sigma : s_0 s_1 s_2 \ldots s_{n+1} \ldots$ satisfying a finite word $w = b_0 b_1 \ldots b_n$ is said to be *accepting* w iff s_{n+1} satisfies $\mathcal{A}$. An infinite run $\sigma : s_0 s_1 s_2 \ldots$ is said to be *satisfying an infinite word* $w = b_0 b_1 \ldots$ over B_U iff for every $i \geq 0$, $s_i \models b_i$.

Discrete transition systems can be composed in parallel. Let $\mathcal{D}_i = \langle V_i, \Theta_i, \rho_i, \mathcal{A}_i \rangle$, $i \in \{1, 2\}$, be two discrete transition systems. We denote the *synchronous parallel composition* of $\mathcal{D}_1$ and $\mathcal{D}_2$ by $\mathcal{D}_1 \,|||\, \mathcal{D}_2$ and define it to be $\mathcal{D}_1 \,|||\, \mathcal{D}_2 = \langle V_1 \cup V_2,\ \Theta_1 \wedge \Theta_2,\ \rho_1 \wedge \rho_2,\ \mathcal{A}_1 \wedge \mathcal{A}_2 \rangle$. We can view the execution of $\mathcal{D}$ as the *joint* execution of $\mathcal{D}_1$ and $\mathcal{D}_2$.

From finite automata to DTS

Given a non-deterministic finite automata on finite words (NFA) [10] whose alphabet is a set of boolean expressions over a given set of variables V, it is straightforward to construct the DTS corresponding to it.

Formally, let V be a set of state variables and let B be the corresponding set of boolean expressions. Let $N = \langle B, Q, Q_0, \delta, A \rangle$ be an NFA. Let *state* be a new variable (not in V) whose domain is Q. Then, N can be represented as the DTS $\mathcal{D}_N = \langle V_N, \Theta_N, \rho_N, \mathcal{A}_N \rangle$ where

$$V_N = V \cup \{state\}; \quad \Theta_N = \bigvee_{q_0 \in Q_0} (state = q_0); \quad \mathcal{A}_N = \bigvee_{q \in A} (state = q); \text{ and}$$

$$\rho_N = \bigvee_{(q_1, b, q_2) \in \delta} (state = q_1 \wedge b \wedge state' = q_2).$$

Let $\sigma = s_0 s_1 \ldots$ be a run of $\mathcal{D}_N$. We say that the "step" (s_i, s_{i+1}) of $\mathcal{D}_N$ corresponds to the transition $(q_1, b, q_2) \in \delta$ of N iff $(s_i, s_{i+1}) \models (state = q_1 \wedge b \wedge state' = q_2)$.

Example 3. Assume we have the set of state variables $V = \{a, b, c\}$. Let N be the NFA over B_V described in Fig. 1. Then the DTS $\mathcal{D}_N = \langle V_N, \Theta_N, \rho_N, \mathcal{A}_N \rangle$ described below corresponds to N, where st_N is a new variable with domain $\{1, 2, 3\}$.

$$V_N = \{a, b, c, st_N\}; \quad \Theta_N = (st_N = 1); \quad \mathcal{A}_N = (st_N = 3); \quad \text{and}$$
$$\rho_N = \begin{array}{l} (st_N = 1 \wedge a \wedge st'_N = 2) \vee \\ (st_N = 2 \wedge b \wedge st'_N = 2) \vee \\ (st_N = 2 \wedge c \wedge st'_N = 3). \end{array}$$

2.2. The logic

The logic considered in this paper is the fragment of the IEEE standard temporal logic PSL [11] which consists of only not r! formulas, with r being a Semi-extended regular expression (SERE). These formulas are of special interest for several reasons. First, it is a convenient way for specification which is widely used. Second, a large subset of the PSL properties can be automatically translated into not r! properties [5]. Finally, the verification of these properties can be reduced to verification of invariant properties and is thus very efficient [5].

The semantics of these formulas over a given DTS is defined below. The definition assumes a set of state variables V, the corresponding set Σ of interpretations of the state variables in V and the set B of boolean expressions over V. We assume two designated boolean expressions *true* and *false* belong to B, such that for every $s \in \Sigma$, $s \models true$ and $s \not\models false$.

Definition 4 (SEREs). • The empty set $\emptyset$ and the empty regular expression λ are SEREs.
- Every boolean expression $b \in B$ is a SERE.
- If r, r_1, and r_2 are SEREs, then the following are also SEREs:
  1. $\{r\}$ (encapsulation) 2. $r_1 \cup r_2$ (union) 3. $r_1 \cdot r_2$ (concatenation)
  4. r^* (Kleene closure) 5. $r_1 \circ r_2$ (fusion) 6. $r_1 \cap r_2$ (intersection).

We denote by RES standard regular expressions, i.e. the subset of SEREs with no fusion or intersection operators. To define the semantics of SEREs, we use the following notations.

Notations

We denote a letter from Σ by s (possibly with subscripts) and a word from Σ by u, v, or w. The *concatenation* of u and v is denoted by uv. If u is infinite, then $uv = u$. The empty word is denoted by ϵ, so that $w\epsilon = \epsilon w = w$. We denote the length of a word w by $|w|$. The empty word $w = \epsilon$ has length 0, a finite word $w = (s_0 s_1 s_2 \cdots s_n)$ has length $n + 1$, and an infinite word has length ∞. We use i, j, and k to denote non-negative integers. For $i < |w|$ we use w^i to denote the $(i + 1)$th letter of w (since counting of letters starts at zero), and we denote by $w^{i..}$ the suffix of w starting at w^i.

The *overlapping concatenation* of us and sv, denoted by $us \circ sv$, is the word usv. Let L_1 and L_2 be sets of words. The *concatenation* of L_1 and L_2, denoted by $L_1 L_2$ is the set $\{w \mid \exists w_1 \in L_1,\ \exists w_2 \in L_2 \text{ and } w = w_1 w_2\}$. The *overlapping concatenation*

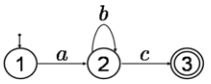

Fig. 1. An NFA over B_V where $V = \{a, b, c\}$.

of L_1 and L_2, denoted by $L_1 \circ L_2$ is the set $\{w \mid \exists w_1 s \in L_1,\ \exists s w_2 \in L_2 \text{ and } w = w_1 s w_2\}$. Define $L^0 = \{\epsilon\}$ and $L^i = LL^{i-1}$ for $i \geq 1$. The *Kleene closure* of L denoted by L^* is the set $\bigcup_{0 \leq i < \omega} L^i$.[1] The notation L^+ is used for the set $\bigcup_{0 < i < \omega} L^i$.

For a formula φ and a sub-formula of it ψ, we denote by $\varphi[\psi \leftarrow \psi']$ the formula obtained by replacing every occurrence of ψ in φ by ψ'.

Definition 5 (SEREs *semantics*). The semantics of SEREs are defined using the relation $\models$ between SEREs over B and finite words over Σ. When $w \models r$ we say that word w tightly satisfies SERE r. The semantics of SEREs are defined as follows, where w is a finite (possibly empty) word over Σ.

- $w \not\models \emptyset$ and $w \models \lambda \iff w = \epsilon$.
- Let b denote a boolean expression in B. Then, $w \models b \iff |w| = 1$ and $w^0 \models b$.
- Let r, r_1, and r_2 denote SEREs over B. Then,

(1) $w \models \{r\} \iff w \models r$

(2) $w \models r_1 \cup r_2 \iff w \models r_1$ or $w \models r_2$

(3) $w \models r_1 \cdot r_2 \iff \exists w_1, w_2$ s.t. $w = w_1 w_2$, $w_1 \models r_1$, and $w_2 \models r_2$

(4) $w \models r^* \iff w = \epsilon$ or $\exists w_1, w_2$ s.t. $w_2 \neq \epsilon$, $w = w_1 w_2$, $w_1 \models r^*$ and $w_2 \models r$

(5) $w \models r_1 \circ r_2 \iff \exists w_1, w_2$, and s s.t. $w = w_1 s w_2$, $w_1 s \models r_1$, and $s w_2 \models r_2$

(6) $w \models r_1 \cap r_2 \iff w \models r_1$ and $w \models r_2$.

We note that the semantics given here are a bit different than the traditional semantics given to regular expressions (and their extensions). The traditional semantics, repeated below, defines a set of words $\mathbb{L}(r)$ satisfying a regular expression. This set consists of words over the same alphabet as the regular expressions themselves. On the other hand, the semantics in Definition 5 relate regular expressions over one alphabet B (the set of boolean expressions over the state variables V) to words over another alphabet Σ (the set of interpretations of the state variables V).

Definition 6 (*The Language of* SEREs). Let Γ be a finite set of symbols (an alphabet). Let b be a letter in Γ and r, r_1, and r_2 SEREs over Γ. The set $\mathbb{L}(r)$, defined below, denotes the set of words over Γ satisfying r according to the traditional semantics of regular expressions.

- $\mathbb{L}(\emptyset) = \emptyset$
- $\mathbb{L}(\lambda) = \{\epsilon\}$
- $\mathbb{L}(b) = \{b\}$
- $\mathbb{L}(\{r\}) = \mathbb{L}(r)$
- $\mathbb{L}(r_1 \cup r_2) = \mathbb{L}(r_1) \cup \mathbb{L}(r_2)$
- $\mathbb{L}(r_1 \cdot r_2) = \mathbb{L}(r_1)\mathbb{L}(r_2)$
- $\mathbb{L}(r^*) = \mathbb{L}(r)^*$
- $\mathbb{L}(r_1 \circ r_2) = \mathbb{L}(r_1) \circ \mathbb{L}(r_2)$
- $\mathbb{L}(r_1 \cap r_2) = \mathbb{L}(r_1) \cap \mathbb{L}(r_2)$.

Example 7. According to Definition 5 the SERE $a \cdot b^* \cdot c$ is satisfied over any word over Σ whose first letter interprets a as *true*, whose last letter interprets c as *true* and any letter in between interprets b as true; among which is for example the word $w = s_0 s_1 s_2 s_3$ with $s_0 = \{a, c\}$, $s_1 = \{b\}$, $s_2 = \{b, c\}$ and $s_3 = \{a, c\}$.

According to Definition 6, $\mathbb{L}(a \cdot b^* \cdot c)$ consists of the set of words over B starting with the letter a followed by a finite number of letters b and ending with the letter c. Thus, the NFA in Fig. 1 accepts $\mathbb{L}(a \cdot b^* \cdot c)$.

Another difference between the traditional semantics and the one given here should be noted. While in the traditional semantics *letters* of the alphabet are mutually exclusive, this is not the case for our semantics. Since in our semantics the letters are actually Boolean expressions, two different letters may hold simultaneously.

Interpreting formulas of the logic over the computational model

Definition 8. Let $\mathcal{D}$ be a discrete transition system, and r a SERE such that $\epsilon \not\models r$. We say that $\mathcal{D}$ *satisfies* the formula not r!, denoted by $\mathcal{D} \models$ not r!, iff for all finite runs σ of $\mathcal{D}$, $\sigma \not\models r$.

We use the syntax not r! to be compliant with PSL [11]. The semantics given here to not r! are equivalent to the ones given in PSL for negating a strong SERE.[2]

To verify a not r! formula, we can run a DTS representing an NFA accepting r in parallel with the given model, and check that the joint run does not reach a finite accepting state of $\mathcal{D}_r$.

[1] Here ω denotes the first transfinite ordinal number.

[2] Note that never r! in PSL is equivalent to not $\{true^* \cdot r\}$!.

Proposition 9. *Let $\mathcal{D}_M$ be a* DTS *and r an* RE. *Let $\mathcal{D}_r$ be the* DTS *representation of an* NFA *accepting $\mathbb{L}(r)$. Let $\mathcal{A}_r$ be the finite acceptance condition of the corresponding* DTS, *$\mathcal{D}_r$. Then,*

$$\mathcal{D}_M \models \text{not } r! \iff \mathcal{D}_M \,|||\, \mathcal{D}_r \models AG\,\neg\mathcal{A}_r.$$

This proposition asserts that verification of not r! properties can be done on-the-fly (see [7]). That is, verification can be done during reachability analysis, without necessarily building the entire model.

Example 10. As stated in Example 7 the NFA of Fig. 1 accepts the SERE $\{a{\cdot}b^*{\cdot}c\}$. By Proposition 9 verifying not $\{a{\cdot}b^*{\cdot}c\}$! over a model $\mathcal{D}_M$ reduces to verifying AG $\neg(st_N = 3)$ over the parallel composition of D_M with the DTS $\mathcal{D}_N$ of Example 3.

3. Embedding a DTS into an RE

In this section we provide an algorithm to embed an automaton into a SERE. Let r be a SERE, and let t be a sub-SERE of r. We construct an independent side-automaton (a DTS to be exact) for t. We replace the occurrence of t inside r with a *placeholder* to get $r' = r[t \leftarrow \mathit{placeholder}]$. We then construct a DTS for r' and derive the invariant formula in the usual way (see Proposition 9). Finally, we run the two DTSs in parallel with the model. We prove that this method gives equivalent results to the one without the embedding.

The idea behind the procedure can be explained as follows. Assume we have a side-automaton for t. In order to embed a placeholder for t in r we need the DTS to start running at the place of embedding. That is, if $r = \{a{\cdot}b{\cdot}\{t\}{\cdot}c\}$, we would like the side-automaton to start running after two time units. If, for example, $r = \{a{\cdot}b^*{\cdot}\{t\}{\cdot}c\}$, then we need the side-automaton to start running at many possible time units: after one 'a' and any finite number of 'b's. Furthermore, we want the side-automaton to be independent of the SERE in which it is embedded, in order to allow the reuse of it. We therefore let it start non-deterministically at any point of time. This is done by adding an idle state with a self-loop, which can transit to the initial states at any time. The placeholder inserted instead of t, will then make sure we check our r' only on those paths where the side-automaton indeed starts when it is expected to.

Since the sub-SERE can be embedded in the RE more than once (e.g. $r = \{a{\cdot}\{t\}{\cdot}b^*{\cdot}\ \{t\}{\cdot}c\}$), or within a starred sub-SERE of r (e.g. $r = \{a{\cdot}\{b{\cdot}\{t\}\}^*{\cdot}c\}$), we need to allow the side-automaton to start running again after it completed a run. To achieve this, we add a transition from the predecessors of final states to the initial states. The result is a special NFA which we call a *cruising* NFA.

We observe that since several automata can be plugged into an RE, and since the automata may be non-deterministic, the intermediate representation combines regular expressions with non-deterministic automata with both a non-determinism and a concurrency component, and is thus doubly exponentially more succinct then regular expressions [8].

The rest of this section is organized as follows. In Section 3.1 below we formally define the cruising NFA, and show that such an NFA can be constructed for any given SERE r. In Section 3.2 we give the definition of the "placeholder" which replaces a sub-SERE, and in Section 3.3 we formally state that the resulting system consisting of the new SERE and side-automaton accepts the same language as the original one, and give examples. In Section 3.4 we prove this result.

3.1. Cruising automaton

Definition 11 (*Cruising* NFA). An NFA $N = \langle B, Q, Q_0, \delta, A\rangle$ is *cruising* iff the following conditions hold:

(1) Initial stuttering: There exists an initial state $q_I \in Q_0$ such that $\forall q_0 \in Q_0$ there exists a δ-transition $(q_I, \mathit{true}, q_0)$. In the sequel we refer to q_I as the *idle* state.
(2) Distinguished start and repetition: $Q_0 \setminus \{q_I\} \neq \emptyset$ and $\forall q_0 \in Q_0$ there exists a δ-transition (q, b, q_0) with $q \neq q_I$ iff there exists a δ-transition (q, b, q_A) to some accepting state $q_A \in Q_A$.

For any SERE r one can build a cruising NFA for r, as stated in the next proposition. Note that the cruising NFA accepts $\mathbb{L}(\{\mathit{true}^*{\cdot}r\}^+)$ rather than $\mathbb{L}(r)$.

Proposition 12. *Let r be an* RE. *There exists a cruising* NFA *accepting $\mathbb{L}(\{\mathit{true}^*{\cdot}r\}^+)$.*

Proof Sketch. Let $N_r = \langle B, Q, Q_0, \delta, A\rangle$ be an NFA for r. We construct a cruising NFA $N'_r = \langle B, Q', Q'_0, \delta', A\rangle$ from N_r as follows. The idea is to add two states: an *idle state* q_I and a *distinguished start state* q_S. Both are initial states. The idle state has a self-loop with the condition *true* (meaning every letter) and a transition to state q_S, again with condition *true*. The state q_S has transitions to all states that are reachable from some initial state of the original automaton N_r, thus making up for the fact that these states are no longer initial. In addition for any transition (q, b, q_A) to a final state of N_r we add transitions (q, b, q_0) to both q_S and q_I. This is done to adhere to the requirement of repetition. The formal definition of N'_r follows.

- $Q' = Q \cup \{q_S, q_I\}$ where $q_S, q_I \notin Q$
- $Q'_0 = \{q_S, q_I\}$
- $\delta' = \delta \cup (q_I, \mathit{true}, q_I) \cup (q_I, \mathit{true}, q_S) \cup$
  $\{(q_S, b, q) \mid q \in Q \text{ and } \exists q_0 \in Q_0, b \in B, \text{ such that } (q_0, b, q) \in \delta\} \cup$
  $\{(q, b, q_0) \mid q \in Q,\ b \in B,\ q_0 \in Q'_0, \text{ and } \exists q_A \in A \text{ s.t. } (q, b, q_A) \in \delta\}.$

Then N'_r is a cruising NFA for r. The correctness of the construction is given in the Appendix. □

Example 13. Applying the construction given in the proof of Proposition 12 on the NFA of Fig. 1 results in the NFA shown in Fig. 2. Note that state 1 is redundant. This does not harm the correctness of the algorithm. In practice, a stage removing such redundant states is added to the procedure, to generate smaller NFAs.

3.2. Defining the placeholder

By using the DTS representation of a cruising NFA N we can construct Boolean expressions which refer to the states of N. We use such Boolean expressions to construct the "placeholder" to replace t in r. We define three conditions which state whether the DTS is in the start, middle or end of its computation. Formally,

Definition 14 (*start, middle, end*). Let $N = \langle B, Q, Q_0, \delta, A\rangle$ be a cruising NFA with idle state q_I. Let $\mathcal{D}_N$ be its DTS representation. Define new boolean expressions start, middle and end over the variables of V and the variable *state* representing the state of N, as follows:

$$\bullet\ \text{start} : \bigvee_{\{(q_0,b,q)\in\delta \mid q_0\in Q_0\setminus\{q_I\}\}} (state = q_0)\wedge b \quad \bullet\ \text{middle} : \neg\text{start} \quad \bullet\ \text{end} : \bigvee_{\{(q,b,q_A)\in\delta \mid q_A\in A\}} (state = q)\wedge b.$$

Example 15. For the NFA described in Fig. 2, Definition 14 gives:

$$\bullet\ \text{start} : (st_N = q_S)\wedge a \quad \bullet\ \text{middle} : (st_N \neq q_S)\vee\neg a \quad \bullet\ \text{end} : (st_N = 2)\wedge c.$$

We are now ready to define the placeholder for a given sub-SERE. Note that the definition of start, middle and end may change for different applications. We therefore give them as parameters to the placeholder function.

Definition 16. Let r be a SERE and t be a sub-SERE of r. Let $\mathcal{D}_N$ be a cruising DTS for t. Let start, middle, end be boolean expressions. Define the SERE $t' = placeholder(t, \text{start}, \text{middle}, \text{end})$ as follows[3]:

$$t' = \begin{cases} (\text{start}\wedge\text{end})\cup\{\text{start}\cdot\text{middle}^*\cdot\text{end}\} & \text{if } \epsilon\notin\mathbb{L}(t) \\ \lambda\cup(\text{start}\wedge\text{end})\cup\{\text{start}\cdot\text{middle}^*\cdot\text{end}\} & \text{Otherwise}. \end{cases}$$

Example 17. Applying Definition 16 to Example 15 we obtain the following definition for the SERE defining the placeholder t' for $\{a\cdot b^*\cdot c\}$.

$$t' := \{((st_N = q_S)\wedge a)\cdot((st_N \neq q_S)\vee\neg a)^*\cdot((st_N = 2)\wedge c)\}.$$

Note that the first disjunct $((st_N = q_S)\wedge a)\wedge((st_N = 2)\wedge c)$ is equivalent to *false* and is therefore omitted.

3.3. Main result

We claim that if we let $\mathcal{D}_{N'}$ run in parallel with the model, then t' can be "plugged" instead of t in almost any SERE r containing t as a sub-SERE. The only cases where this claim does not hold, are when t appears in both operands of the intersection or fusion operators (since these operators have a parallel nature). Formally, we claim the following.

Theorem 18. *Let $\mathcal{D}_M$ be a* DTS, *r a* SERE *and t a sub-*SERE *of r such that for every intersection and fusion operator, t is not a sub-*SERE *of both operands. Let* start, middle, end *be as in Definition 14. Let the* DTS $\mathcal{D}_t$ *be a cruising* DTS *for t. Then,*

$$\mathcal{D}_M \models \text{not } r! \iff \mathcal{D}_M \,|||\, \mathcal{D}_t \models \text{not } r[t \leftarrow placeholder(t, \text{start}, \text{middle}, \text{end})]!$$

Example 19. According to Theorem 18 verifying not $\{d\cdot e^*\cdot a\cdot b^*\cdot c\cdot f\}$! over a given model $\mathcal{D}_M$ is equivalent to verifying not $\{d\cdot e^*\cdot t'\cdot f\}$! (with t' as defined in Example 17) over the parallel composition of the model $\mathcal{D}_M$ with the model $\mathcal{D}_N$ of Example 3.

Corollary 20. *Let $\mathcal{D}_M$ be a* DTS, *r a* SERE *and t a sub-*SERE *of r such that for every intersection and fusion operator, t is not a sub-*SERE *of both operands. Let the* DTS $\mathcal{D}_t$ *be a cruising* DTS *for t. Finally, let t' be the placeholder defined above and let r' be $r[t \leftarrow t']$. Then,*

$$\mathcal{D}_M \models \text{not } r! \iff \mathcal{D}_M \,|||\, \mathcal{D}_t \,|||\, \mathcal{D}_{r'} \models AG\,\neg\mathcal{A}_{r'}.$$

Example 21. Let r be the SERE $\{d\cdot e^*\cdot a\cdot b^*\cdot c\cdot f\}$ and t the SERE $a\cdot b^*\cdot c$. Then, $r' = \{d\cdot e^*\cdot\text{start}\cdot\text{middle}^*\cdot\text{end}\cdot f\}$ where start, middle and end are as defined in Example 15. Using only the original state variables $\{a, b, c, d, e, f, st_N\}$ we get $r' = \{d\cdot e^*\cdot((st_N = q_S)\wedge a)\cdot((st_N \neq q_S)\vee\neg a)^*\cdot((st_N = 2)\wedge c)\cdot f\}$. The NFA $N_{r'}$ for r' is described in Fig. 3.

[3] An inductive way to check whether $\epsilon\in\mathbb{L}(t)$ is given in [6] by means of the predicate $\mathcal{S}(r)$.

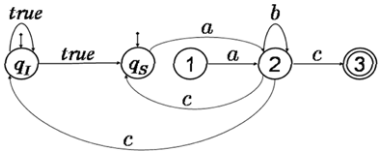

Fig. 2. A cruising NFA accepting the set $\mathbb{L}(\{true^* \cdot \{a \cdot b^* \cdot c\}\}^+)$.

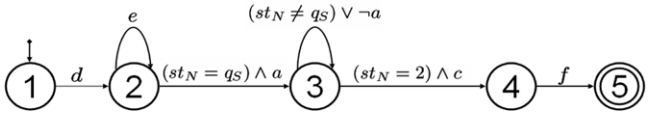

Fig. 3. An NFA accepting the set $\mathbb{L}(r')$ as in Example 21.

The DTS $\mathcal{D}_{r'} = \langle V_{r'}, \Theta_{r'}, \rho_{r'}, \mathcal{A}_{r'} \rangle$ described below corresponds to N'_r, where *state* is a new variable with domain $\{1, 2, 3, 4, 5\}$.

$$V_{r'} = \{a, b, c, d, e, f, st_N, state\}; \quad \Theta_{r'} = (state = 1); \quad \mathcal{A}_{r'} = (state = 5);$$
$$\begin{aligned} \rho_{r'} = \quad & (state = 1 \wedge d \wedge state' = 2) \bigvee (state = 2 \wedge e \wedge state' = 2) \bigvee \\ & (state = 2 \wedge ((st_N = q_S) \wedge a) \wedge state' = 3) \bigvee \\ & (state = 3 \wedge ((st_N \neq q_S) \vee \neg a) \wedge state' = 3) \bigvee \\ & (state = 3 \wedge ((st_N = 2) \wedge c) \wedge state' = 4) \bigvee \\ & (state = 4 \wedge f \wedge state' = 5) \bigvee (state = 5 \wedge state' = 5). \end{aligned}$$

Thus, according to Corollary 20 verifying $\{d \cdot e^* \cdot a \cdot b^* \cdot c \cdot f\}!$ over a given model $\mathcal{D}_M$ is equivalent to verifying AG $\neg(state = 5)$ over the parallel composition of the model $\mathcal{D}_M$ with the model $\mathcal{D}_N$ of Example 3 and the DTS $D_{r'}$ defined above.

Note that information about the embedded SERE can be used to simplify the placeholder and thus reduce the overall size of the automata. For example, when the SERE has no computations of length one it is possible to remove the disjunct start $\wedge$ end from the definition of the placeholder. Similarly, information about the embedding SEREs can be used to simplify the cruising DTS. For example, if the SERE is not embedded more than once nor within a starred sub-SERE, it is possible to build a side-automaton which does not confirm to the condition of repetition.

3.4. Proof of the theorem

We use the following notations and assumptions in the proof of Theorem 18 and its lemmas.

Let r be a SERE over B, and t a sub-SERE of r such that for every intersection and fusion operator, t is not a sub-SERE of both operands. Let $N = \langle B, Q, Q_0, \delta, A \rangle$ be an NFA for t and $N' = \langle B, Q', Q'_0, \delta', A' \rangle$ the cruising NFA constructed from N (as defined in Proposition 12). Let $\mathcal{D}_{N'}$ be the DTS of N' and let start and end be the boolean expressions defined in Definition 14. Let t' be the corresponding placeholder. Define idle to be the boolean expression asserting $state = q_I$.

We use s (possibly with subscripts) to denote states/boolean expressions in a run of a DTS. We use σ (possibly with subscripts) to denote sequences of states/boolean expressions in a run of a DTS. We use b (possibly with subscripts) to denote letters/boolean expressions and w (possibly with subscripts) to denote words i.e. sequences of boolean expressions.

Let r''' denote the SERE (idle $\vee$ start) $\circ$ $r[t \leftarrow t']$ $\circ$ (end $\vee$ idle). Let r'' denote the SERE r''' if $\epsilon \notin \mathbb{L}(r)$ and the SERE $\lambda \cup r'''$ otherwise.

The proof of Theorem 18 makes use of the following 3 lemmas.

Lemma 22. *$w \models r \iff$ there exists a run of $\mathcal{D}_{N'}$ satisfying w which satisfies also r''.*

Proof Sketch. The proof is by induction on the structure of the formula, making use of the following two lemmas.

Lemma 23. *Let σ'_1, σ'_2 be runs of $D_{N'}$ satisfying (idle $\vee$ start) $\circ$ true* $\circ$ (end $\vee$ idle) as well as w_1 and w_2, respectively. Then $\sigma'_1\sigma'_2$ is a run of $\mathcal{D}_{N'}$ satisfying (idle $\vee$ start) $\circ$ true* $\circ$ (end $\vee$ idle) as well as w_1w_2.*

Lemma 24. *Let $\sigma'_1 s_1$ be a run of $D_{N'}$ satisfying (idle $\vee$ start) $\circ$ true* $\circ$ (end $\vee$ idle) as well as $w_1 b$. Let $s_2\sigma'_2$ be a run of $D_{N'}$ satisfying (idle$*$) as well as bw_2. Then $\sigma'_1 s_1 \sigma'_2$ is a run of $\mathcal{D}_{N'}$ satisfying (idle $\vee$ start) $\circ$ true* $\circ$ (end $\vee$ idle) as well as $w_1 bw_2$.*

Let $\sigma'_1 s_1$ be a run of $D_{N'}$ satisfying (idle$*$) as well as $w_1 b$. Let $s_2\sigma'_2$ be a run of $D_{N'}$ satisfying (idle $\vee$ start) $\circ$ true* $\circ$ (end $\vee$ idle) as well as bw_2. Then $\sigma'_1 s_2 \sigma'_2$ is a run of $\mathcal{D}_{N'}$ satisfying (idle $\vee$ start) $\circ$ true* $\circ$ (end $\vee$ idle) as well as $w_1 bw_2$.

The complete proof is given in the Appendix. □

Proof of Theorem 18. $\mathcal{D}_M \not\models$ not $r!$

iff There exists a finite run w of $\mathcal{D}_M$ s.t. $w \models r$

iff [By Lemma 22]
There exists a finite run w of $\mathcal{D}_M$ s.t. there exists a run of $\mathcal{D}_{N'}$ satisfying w which also satisfies r''
iff [By definition of r'']
There exists a finite run w of $\mathcal{D}_M$ s.t. there exists a run of $\mathcal{D}_{N'}$ satisfying w which also satisfies $r[t \leftarrow t']$
iff $\mathcal{D}_M \,|||\, \mathcal{D}_{N'} \not\models \text{not } r[t \leftarrow t']!$ □

In the next section we demonstrate the benefits of using the embedding method, and give several applications.

4. Benefits of the embedding algorithm

We use the embedding algorithm in several different applications, each of which benefits from the use of it in a different way. In Section 4.1 we show how *modularity* is achieved for the *named sequence* construct of PSL, and discuss the implementation of SERE intersection. In Section 4.2 we demonstrate how the *size* of the automaton built for the formula can be reduced when using embedding, and in Section 4.3 we show how *expressiveness* is gained when SEREs are to be translated to CTL.

4.1. Modularity and reuse

As an example to code reuse, consider the *named sequence* mechanism of PSL [11]. In this language, a SERE can be given a name, and then be used as a sub-SERE in many formulas. For example, one can define: sequence $r := \{a{\cdot}b^*{\cdot}c\}$; and then use it in other formulas: not $\{d^*{\cdot}r\}!$, not $\{a{\cdot}r{\cdot}f\}!$ etc. When constructing DTSs for the above formulas, 3 states would be produced for each occurrence of r. Using the embedding mechanism, we can construct a DTS for r only once, and reuse it as many times as needed.

Another example is the implementation of SERE intersection (see Section 2.2). Unwrapping an intersection operation at the SERE level is very complicated [2]. Instead, we use the embedding algorithm to implement intersection. By the general embedding method, we can construct an NFA for any sub-SERE of the form $r_1 \cap r_2$ and plug the placeholder for its DTS instead. An NFA for $r_1 \cap r_2$ can be built using the product construction [10]. However, in order to simplify the construction, the use of embedding for intersection can be extended even further. We provide a method to embed *two* DTS, one for each operand of the intersection, replacing them with a single placeholder which serves for both of them together. This allows us to implement intersection using the same type of DTS, making it easier to maintain.

Formally, let $r_1 \cap r_2$ be a sub-SERE of r. Let N'_1 and N'_2 be cruising NFA for r_1 and r_2 respectively, constructed as in Section 3. Let $D_{N'_1}$ and $D_{N'_2}$ be their DTS representations. Define start_i, end_i for $i \in \{1, 2\}$ as the respective Boolean expressions stating that $D_{N'_i}$ is in the real start (resp. end) of its run (exactly as in the general embedding method). Define the new Boolean expressions start and end as the conjunctions $\text{start}_1 \wedge \text{start}_2$, and $\text{end}_1 \wedge \text{end}_2$, respectively. Define the new boolean expression middle as the conjunction $\neg\text{start}_1 \wedge \neg\text{start}_2$. We claim that the placeholder defined using the above start, middle and end can be plugged instead of $r_1 \cap r_2$ while running the model in parallel with both $D_{N'_1}$ and $D_{N'_2}$.

Proposition 25. *Let $\mathcal{D}_M$ be a* DTS, *r a* SERE *and $r_1 \cap r_2$ a sub-*SERE *of r which is not a sub-*SERE *of both operands of $\circ$ or $\cap$. Let N'_1 and N'_2 be the cruising* NFAs *constructed for r_1 and r_2, respectively. Let* start, middle *and* end *be as defined above.*

$$\mathcal{D}_M \models \text{not } r!$$
$$\Updownarrow$$
$$\mathcal{D}_M \,|||\, \mathcal{D}_{N'_1} \,|||\, \mathcal{D}_{N'_2} \models \text{not } r[r_1 \cap r_2 \leftarrow \mathit{placeholder}(r_1 \cap r_2, \text{start}, \text{middle}, \text{end})]!$$

Proof Sketch. The proof is similar to the proof of Theorem 18, this time making use of the following lemma.

Lemma 26. *Let r be a* SERE, *and $t = r_1 \cap r_2$ a sub-*SERE *of r. Let N'_1 and N'_2 be the cruising* NFAs *for r_1 and r_2 respectively. Let $\mathcal{D}_{N'_1}$ and $\mathcal{D}_{N'_1}$ be their corresponding* DTSs.

Let idle_1 and idle_2 denote, respectively, that $D_{N'_1}$ and $D_{N'_2}$ are in a state s such that $s[\mathit{state}] = q_I$. Let idle denote the conjunction $\mathit{idle}_1 \wedge \mathit{idle}_2$. Let t' denote the SERE *$\mathit{placeholder}(r_1 \cap r_2, \text{start}, \text{middle}, \text{end})$. Let r''' denote the* SERE *$(\mathit{idle} \vee \text{start}) \circ r[t \leftarrow t'] \circ (\text{end} \vee \mathit{idle})$ and let r'' denote the* SERE *r''' if $\epsilon \notin \mathbb{L}(r)$ and the* SERE *$\lambda \cup r'''$ otherwise. Then*

$$w \models\!\!\models r$$
$$\Updownarrow$$

there exists a run of $\mathcal{D}_{N'_1} \,|||\, \mathcal{D}_{N'_2}$ satisfying w which satisfies also r''.

The complete proof is given in the Appendix. □

Table 1
Named sequences: with vs. without using the embedding algorithm

Rule			Non-embedding		Embedding	
#form	#occr	Size	States	Time	States	Time
2	2	90	380	151	122	14
2	2	153	632	192	185	17
3	5	19	247	154	106	14
2	3	32	212	147	70	15
3	5	32	377	1138	119	18

4.2. Efficiency of verification

In some cases, the use of embedding allows us to build a smaller automaton for a given formula, thus reducing the size of the model to be verified.

We give two examples for that. The first is the named sequence mechanism of PSL, described in Section 4.1. When embedding is used to implement a named sequence r, the side-automaton built for r can be plugged into other formulas, or even in different places in the same formula. When many formulas use the same side-automaton in one run, this can bring a significant reduction in the size of the overall automata built for verification. We demonstrate this in Table 1. In this table, *#form* refers to the number of formulas that use the same named sequence, and *#occr* indicates the total number of occurrences of this sequence in the formulas. *Size* is the number of states of the side-automaton built for the named sequence. For each set of formulas (rule) we compare two runs. The first, with no embedding, simply integrates a named sequence into a formula as many times as needed, and then builds one automaton for each formula. The second uses embedding, thus builds one side-automaton for the named sequence, and embeds the appropriate placeholder where needed. The *states* parameter for each run indicates the overall size of the automata built for each rule, and *time* is given in seconds. The experiments were run on Dual 2.4 Ghz Pentium 4 (xeon) with 2.4GB RAM. Table 1 shows a clear advantage for the embedding method over the traditional one.

Our second example is the operator *counters* ($b[= i..j]$). *Counters* can be viewed as abbreviations of the basic SERE operators as follows, where b denotes a boolean expression, i and j denote integer constants such that $i \geq 0$ and $j \geq i$.

$$\bullet\ b[= i] \stackrel{\text{def}}{=} \overbrace{\{\neg b^* \cdot b\} \cdot \ldots \cdot \{\neg b^* \cdot b\}}^{i \text{ times}} \cdot \neg b^* \qquad \bullet\ b[= i..j] \stackrel{\text{def}}{=} b[= i] \cup \ldots \cup b[= j].$$

A straightforward unwrapping of a counter operator $b[= i..j]$ at the SERE level, using the definition above, will result in a SERE of size quadratic in j. Using other SERE-level unwrapping methods we can reduce the size of the resulting SERE. The complexity can be improved to linear by directly building an NFA for a counter. We show how to build an NFA for a counter $b[= i..j]$, which has $j + 2$ states. Given such an NFA, the embedding algorithm can then be used to embed the NFA into the SERE. Below we give the construction of an NFA for a counter, consisting of $j+2$ states. We then present experimental results which compare the two ways of translation.

An NFA *for* $b[= i..j]$

Let b be a boolean expression, i, j integers such that $0 \leq i \leq j$ and $B = \{b, \neg b\}$.

$$\mathbb{L}(b[= i..j]) = \left\{ w \;\middle|\; \begin{array}{l} w \text{ is a word over } B \text{ and if } k \text{ is the number of} \\ \text{letters } \ell \text{ in } w \text{ such that } \ell \models b \text{ then } i \leq k \leq j \end{array} \right\}.$$

The following is an NFA that accepts $\mathbb{L}(b[= i..j])$. The automaton has 'counting' states $q_0, q_1, \ldots, q_{j+1}$. The idea is that state q_k indicates that up until now k b's were read. Thus, all states q_k such that $i \leq k \leq j$ are accepting states.

Formally, define $N_{b[=i..j]} = \langle B, Q, Q_0, \delta, A \rangle$ where

$$Q = \{q_k \mid 0 \leq k \leq j+1\};\ Q_0 = \{q_0\};\ A = \{q_k \mid i \leq k \leq j\};\ \text{and}$$
$$\delta = \ \{(q_k, b, q_{k+1}) \mid 0 \leq k \leq j\} \cup \{(q_k, \neg b, q_k) \mid 0 \leq k \leq j\}.$$

Proposition 27. *The* NFA $N_{b[=i..j]}$ *accepts* $\mathbb{L}(b[= i..j])$.

Proof. Let $w \in \mathbb{L}(\{b[= i..j]\})$, we show there exists a run of $N_{b[=i..j]}$ on w which is accepting. Let k be the number of b's in w. Since $w \in \mathbb{L}(\{b[= i..j]\})$, we have that $i \leq k \leq j$. We can therefore write w as $\neg b^{m^0}, b, \neg b^{m^1}, \ldots, b, \neg b^{m^k}$ where $m^0, \ldots, m^k$ are non-negative integers and $\neg b^{m^i}$ denotes m^i consecutive repetitions of $\neg b$. Thus, by the transition relation $q_0^{m^0+1}, q_1^{m^1+1}, q_2^{m^2+1}, \ldots, q_k^{m^k}$ is an accepting run of $N_{b[=i..j]}$ on w.

Let $w \notin \mathbb{L}(\{b[= i..j]\})$ and assume towards contradiction that $s_0 s_1 \ldots s_n$ is an accepting run of $N_{b[=i..j]}$ on w. Therefore $s_n \in \{q_i, \ldots, q_j\}$. It is easy to see that for all $0 \leq t \leq n$ and $\forall 1 \leq m \leq j$, if $s_t = q_m$ then there are exactly m states in $s_0 s_1 \ldots s_t$ in which b is asserted. Therefore there are between i to j states in $s_0 s_1 \ldots s_n$ in which b is asserted. That is, $w \in \mathbb{L}(\{b[= i..j]\})$, in contradiction to the assumption. □

Table 2
Unwrapped counters vs. embedded counters

Formula	Small model				Large model			
	Non-embedding		Embedding		Non-embedding		Embedding	
	States	Time	States	Time	States	Time	States	Time
$b[= 1..10]$	55	5.15	13	1.25	55	1.3	13	1.25
$b[= 10..20]$	165	8.1	23	1.92	165	1766.03	23	1419.63
$b[= 20..25]$	135	7.85	28	4.8	135	3928.4	28	3188.72
$b[= 25..30]$	165	10.43	33	6.26	165	5105.53	33	3992.37
$b[= 25..40]$	455	60.59	43	6.62	455	12038	43	9397

$N_{b[=i..j]}$ is an NFA for r. Thus, by Proposition 18 we can embed the placeholder for $b[= i..j]$ in any SERE containing it, and run both DTSS in parallel with the model.

We present experimental results of embedding different *counters* formulas. We compare the traditional unwrapping method (see above) to our embedding method, in two aspects: the size of automata built for a formula, and the verification time. The verification time is influenced by the overall size of the model, which includes the design under test as well as the automata built for the formulas. Thus we can see a significant benefit for the embedding method when the design is small, and a smaller improvement when the design under test is large. In Table 2, we present the results of running formulas of the type not $b[= i..j]$! both on a small model of 36 state variables, and on a larger one of 196 state variables (both numbers present the size of the model *after* model reductions). The experiments were done on Dual 2.4 Ghz Pentium 4 (xeon) with 2.4GB RAM.

In this table, the *states* presented are the number of states in the automaton built for the formula, and *time* is given in seconds. As can be seen, the embedding algorithm performs better in all cases. The improvement in performance becomes more significant though, as the formula becomes larger.

4.3. Increasing expressibility

As discussed earlier, not r! formulas are usually verified by constructing an automaton. However, another method exists, which we discuss in this section. This method translates a not r! formula into CTL, so that it can be verified using CTL model checking algorithms.

While the basic operators of regular expressions can be translated into CTL (including concatenation, union, and Kleene closure (a star) on a single letter), it is impossible to translate into CTL a star over a sub-sere (e.g. not $\{a{\cdot}\{b{\cdot}c{\cdot}d\}^*{\cdot}e\}$!), as shown in [18]. This causes a problem in real-life verification: the user writes a formula, and then chooses the model checking algorithm to be applied. However, if he chooses to apply a CTL model checking algorithm and the not SERE! formula contains a starred sub-expression, the tool must reject his choice!

The embedding algorithm can be used to solve this problem. Using the embedding technique, we plug out the starred sub-expression, and plug in the placeholder, as described in Section 3. The formula obtained does not have a starred sere, and can therefore be translated into CTL. The side-automaton is then run in parallel with the model as usual. This combination of a CTL formula with an automaton can be seen as a version of QCTL [13,16], thus augmenting the expressive power of CTL. We present below the translation of a regular expression into CTL.

Definition 28 (*Translation Procedure*). Let r_1, r_2, r be regular expressions, and $b \in B$ be a boolean expression. We define the function T as follows:

- $T(b{\cdot}r) = b \wedge EX(T(r))$
- $T(b^*{\cdot}r) = E\,[b\ U\ T(r)]$
- $T(\{r_1 \cup r_2\}{\cdot}r) = T(r_1{\cdot}r) \vee T(r_2{\cdot}r)$
- $T(b) = b$
- $T(r{\cdot}b^*) = T(r)$
- $T(r_1 \cup r_2) = T(r_1) \vee T(r_2)$.

Note that the function T produces an ECTL formula. In order to get an equivalent to the given not r! formula, we need to negate the output of T and obtain an ACTL formula, as stated in the proposition below.

Proposition 29. *Let $\mathcal{D}_M$ be a discrete system and r a* SERE *with no starred sub-*SEREs*, such that $\epsilon \not\models r$. Then, $\mathcal{D}_M \models$ not r! $\iff$ $\mathcal{D}_M \models \neg T(r)$.*

Proof Sketch. In order to prove this proposition we need to define the semantics of not r! over a state and a model, as done for any CTL formula.

Definition 30. Let $\mathcal{D}_M$ be a DTS and $s \in \Sigma_V$. Let r be a SERE such that $\epsilon \not\models r$.

$\mathcal{D}_M, s \models$ not r! $\iff$ for every finite run σ of $\mathcal{D}_M$ with $\sigma^0 = s$ holds that $\sigma \not\models r$.

We note that the translation procedure (Definition 28) covers all SEREs r such that $\epsilon \not\models r$ and r does not contain any starred sub-SEREs. We also note that Definition 30 is legitimate, since although SEREs are defined over non-empty words, the definition $\mathcal{D}_M \models \text{not } r!$ relies on the fact that runs are by definition non-empty and further assumes $\epsilon \not\models r$. From the same reason we can first show that for every $s \in \Sigma_V$,

$$\mathcal{D}_M, s \models \text{not } r! \iff \mathcal{D}_M, s. \models \neg T(r).$$

This is shown by induction on the structure of r, in the Appendix. □

5. Conclusions

We have revisited the translation of extended regular expressions (SEREs) into an automaton, which is the main translation path of the IBM model checking tool-set RuleBase [3]. We have presented an *embedding* algorithm, that allows an automaton to be embedded into a SERE. Thus, instead of producing one automaton per formula, the translation using embedding may consist of several automata, running in parallel with the model. An important nature of this algorithm is that the auxiliary automata built are independent of the original SERE, and thus can be used more than once.

From a theoretical point of view, our embedding algorithm produces E,C-automata [8], since a set of NFAs are run concurrently with the model. Thus, according to [8], this representation is exponentially more succinct than NFAs. Moreover, we assume an underlying representation that combines RES with E,C-automata. A future work is to formally define this representation and compare its succinctness to that of SEREs.

While for many not SERE! formulas the traditional translation into one automaton is good enough, we have shown various cases that can significantly benefit from using our embedding algorithm. For *named sequences*, the embedding algorithm allows reuse of the automata code. For *intersection*, the use of embedding allows us to enjoy the same NFA construction, avoiding the need to unwrap at the SERE level or to intersect at the automata level. For *counters* and *named sequences* we manage to reduce the size of the automata built for the formulas, and for CTL *translation* we extend the expressive power of the language. The variability of these applications demonstrates its potential, and suggests, we believe, that more applications may be found for it.

Automata for SERE formulas are sometimes used as checkers for simulation [1]. For that purpose, the automaton must be deterministic. Since the embedding algorithm relies strongly on the non-deterministic nature of the side-automaton (the side-automaton can start at any cycle), it should be carefully examined whether the embedding algorithm performs well for simulation checkers, as it does for model checking.

Acknowledgments

We would like to thank Cindy Eisner, Orna Lichtenstein and Avigail Orni for their helpful comments on an early draft of this paper. The work of the second author was carried out at the John von Neumann Minerva Center for the Verification of Reactive Systems.

Appendix

We give here the full proofs of Propositions 12, 25 and 29 and Theorem 18.

A.1. Proof of Proposition 12

Proposition 12. Let r be an RE. There exists a cruising NFA accepting $\mathbb{L}(\{true * r\}^+)$.

Proof. Let $N_r = \langle B, Q, Q_0, \delta, A\rangle$ be an NFA for r. Define $N'_r = \langle B, Q', Q'_0, \delta', A\rangle$ as follows.

- $Q' = Q \cup \{q_S, q_I\}$ where $q_S, q_I \notin Q$
- $Q'_0 = \{q_S, q_I\}$
- $\delta' = \delta \cup (q_I, true, q_I) \cup (q_I, true, q_S) \cup$
  $\{(q_S, b, q) \mid q \in Q \text{ and } \exists\, q_0 \in Q_0,\ b \in B,\ \text{such that } (q_0, b, q) \in \delta\} \cup$
  $\{(q, b, q_0) \mid q \in Q,\ b \in B,\ q_0 \in Q'_0,\ \text{and } \exists q_A \in A \text{ s.t. } (q, b, q_A) \in \delta\}$.

It is easy to see that q_I is the initial stuttering state, q_S is the distinguished start state, and that together with the transition relation the conditions of Definition 11 are satisfied. Therefore N'_r is cruising. It remains to show N'_r accepts $\mathbb{L}(\{true^* \cdot r\}^+)$. Let $w \in \mathbb{L}(\{true^* \cdot r\}^+)$. The proof is by induction on the structure of w. If $w = uv$ where $u \in \mathbb{L}(true*)$ and $v = v^0 \dots v^{n_1-1} \in \mathbb{L}(r)$ then there exists a run

$$q_0 \xrightarrow{v^0} q_1 \xrightarrow{v^1} q_2 \quad \cdots \quad q_{n_1-1} \xrightarrow{v^{n_1-1}} q_{n_1}$$

of N_r accepting v. If $|u| = 0$ then by the transition relation of N'_r

$$q_S \xrightarrow{v^0} q_1 \xrightarrow{v^1} q_2 \quad \cdots \quad q_{n_1-1} \xrightarrow{v^{n_1-1}} q_{n_1}$$

is a run of N'_r accepting w. If $u = true^{m_1}$ where $m_1 \geq 1$

$$\underbrace{q_I \xrightarrow{true} q_I \ldots \xrightarrow{true}}_{m_1 \text{ times}} q_S \xrightarrow{v^0} q_1 \xrightarrow{v^1} q_2 \quad \cdots \quad q_{n_1-1} \xrightarrow{v^{n_1-1}} q_{n_1}$$

is a run of N'_r accepting w. Assume the claim holds for $w_1 \in \mathbb{L}(\{true^* \cdot r\}^k)$ where $k \geq 1$. Let $w = w_1 u v$ where $u \in \mathbb{L}(\{true^*\})$ and $v \in \mathbb{L}(r)$. Let

$$p_0 \xrightarrow{w_1^0} p_1 \xrightarrow{w_1^1} p_2 \quad \cdots \quad p_{n-1} \xrightarrow{w_1^{n-1}} p_n$$

be an accepting run of N'_r over $w_1 = w_1^0 \ldots w_1^{n-1}$ and

$$q_0 \xrightarrow{v^0} q_1 \xrightarrow{v^1} q_2 \quad \cdots \quad q_{n_1-1} \xrightarrow{v^{n_1-1}} q_{n_1}$$

an accepting run of N_r over $v = v^0 \ldots v^{n_1-1}$.

If $|u| = 0$ then

$$p_0 \xrightarrow{w_1^0} p_1 \xrightarrow{w_1^1} p_2 \quad \cdots \quad p_{n-1} \xrightarrow{w_1^{n-1}} q_S \xrightarrow{v^0} q_1 \xrightarrow{v^1} q_2 \quad \cdots \quad q_{n_1-1} \xrightarrow{v^{n_1-1}} q_{n_1}$$

is a run of N'_r accepting w. If $|u| > 0$ and $u = true^{m_1}$

$$p_0 \xrightarrow{w_1^0} p_1 \xrightarrow{w_1^1} p_2 \quad \cdots \quad p_{n_1-1} \xrightarrow{w_1^{n_1-1}} \underbrace{q_I \xrightarrow{true} q_I \ldots q_I \xrightarrow{true}}_{m_1 \text{ times}}$$

$$q_S \xrightarrow{v^0} q_1 \xrightarrow{v^1} q_2 \quad \cdots \quad q_{n-1} \xrightarrow{v^{n_2-1}} q_{n_2}$$

is a run of N'_r accepting w.

For the other direction, assume σ is an accepting run of N'_r over w. Let $pred(A)$ denote the set $\{q \in Q | \exists b \in B, q_A \in Q_A, \text{ s.t. } (q, b, q_A) \in \delta\}$. By the transition relation, σ is of the form $\sigma = \bar{\sigma}_1 \ldots \bar{\sigma}_k$ for some $k \geq 1$ where for every $i \leq k$, $\bar{\sigma}_i$ is in one of the following forms:

(1)

$$q_S \xrightarrow{v_i^0} q_1^i \ldots q_{n_i-1}^i \xrightarrow{v_i^{n_i-1}} q_{n_i}^i$$

(2)

$$\underbrace{q_I \xrightarrow{true} q_I \ldots q_I \xrightarrow{true}}_{m_i \text{ times}} q_S \xrightarrow{v_i^0} q_1^i \ldots q_{n_i-1}^i \xrightarrow{v_i^{n_i-1}} q_{n_i}^i$$

$q_{n_i-1}^i \in pred(A)$, and $q_{n_k} \in A$. The rest of the proof is for case (2). The proof for case (1) is similar. Define

$$w_k = true^{m_1} v_1^0 \ldots v_1^{n_1-1} \ldots u_k^0 true^{m_k} v_k^0 \ldots v_k^{n_k-1}$$

(the word accepted by σ). We show $w_k \in \mathbb{L}(\{true^* \cdot r\}^k)$ by induction on k. If $k = 1$, then

$$\underbrace{q_I \xrightarrow{true} q_I \ldots q_I \xrightarrow{true}}_{m_1 \text{ times}} q_S \xrightarrow{v^0} q_1^1 \ldots q_{n_1-1}^1 \xrightarrow{v^{n_1-1}} q_{n_1}^1$$

and $w_1 = true^{m_1} v^0 \ldots v^{n_1-1}$. By the transition relation $\exists q_0 \in Q_0$ such that $(q_0, v^0, q_1^1) \in \delta$ and $\forall 1 \leq i \leq n_1 - 1$, $(q_i^1, v^i, q_{i+1}^1) \in \delta$ so $q_0, \ldots, q_{n_1}^1$ is an accepting run of N_r over $v^0 \ldots v^{n_1-1}$, therefore $v^0 \ldots v^{n_1-1} \in \mathbb{L}(r)$ and $w_1 \in \mathbb{L}(\{true^* \cdot r\})$. Assume that if $\sigma = \bar{\sigma}_1 \ldots \bar{\sigma}_k$ is an accepting run over w_k then $w_k \in \mathbb{L}(\{true^* \cdot r\}^k)$. Let $\bar{\sigma}_k = \hat{\sigma}_k \xrightarrow{v_k^{n_k-1}} q_{n_k}^k$.That is, $\hat{\sigma}_k$ is the prefix of $\bar{\sigma}_k$ obtained by chopping the last state of $\bar{\sigma}_k$. The last state of $\hat{\sigma}_k$ is $q_{n_k-1}^k \in pred(A)$.

Let

$$\sigma^{k+1} = \bar{\sigma}_1 \ldots \bar{\sigma}_{k-1} \hat{\sigma}_k \xrightarrow{v_k^{n_k-1}} \underbrace{q_I \xrightarrow{true} q_I \ldots q_I \xrightarrow{true}}_{m_{k+1} \text{ times}}$$

$$q_S \xrightarrow{v_{k+1}^0} q_1^{k+1} \ldots q_{n_{k+1}-1}^{k+1} \xrightarrow{v_{k+1}^{n_{k+1}-1}} q_{n_{k+1}}^{k+1}$$

be an accepting run of N'_r over $w_{k+1} = w_k \cdot true^{k+1} v^0_{k+1} \dots v^{n_{k+1}-1}_{k+1}$. $q^k_{n_k-1} \in Q$ and there exists $q \in \{q_I, q_S\}$ such that $(q^k_{n_k-1}, v^{n_k-1}_k, q) \in \delta'$. By the definition of δ', there exists $q^k_{n_k} \in A$ such that $(q^k_{n_k-1}, v^{n_k-1}_k, q^k_{n_k}) \in \delta$. Therefore

$$\sigma' = \bar{\sigma}_1 \dots \bar{\sigma}_{k-1} \hat{\sigma}_k \xrightarrow{v^{n_k-1}_k} q^k_{n_k}$$

is an accepting run of N'_r over w_k. By the induction hypothesis $w_k \in \mathbb{L}(\{true^* r\}^k)$.

$$\underbrace{q_I \xrightarrow{true} q_I \dots q_I \xrightarrow{true}}_{m_{k+1} \text{ times}} q_S \xrightarrow{v^0_{k+1}} q^{k+1}_1 \dots q^{k+1}_{n_{k+1}-1} \xrightarrow{v^{n_{k+1}-1}_{k+1}} q^{k+1}_{n_{k+1}}$$

is an accepting run of N'_r over $u^0_{k+1} u^1_{k+1} \dots u^{m_{k+1}-1}_k v^1_k \dots v^{n_{k+1}-1}_{k+1}$ by the induction hypothesis

$$true^{k+1} \cdot v^0_{k+1} \dots v^{n_{k+1}-1}_{k+1} \in \mathbb{L}(\{true^* r\})$$

and $w \in \mathbb{L}(\{true^* r\}^{k+1})$. □

A.2. Proof of Theorem 18

Lemma 22.

$$w \models\!\models r$$
$$\Updownarrow$$
there exists a run of $\mathcal{D}_{N'}$ satisfying w which satisfies also r''.

Proof. The proof is by induction on the structure of r with respect to t. The base case is $r = t$. The induction step can be decomposed into 7 cases, where r_1 and r_2 are SEREs such that t may be a sub-SERE of them and n_1 and n_2 are SEREs such that t is not a sub-SERE of them:

$$r = r_1 \cdot r_2 \qquad r = r_1 \cup r_2 \qquad r = r_1{}^* \qquad r = r_1 \circ n_2 \qquad r = r_1 \cap n_2$$
$$r = n_1 \circ r_2 \qquad r = n_1 \cap r_2.$$

Denote $r_1[s \leftarrow s']$ and $r_2[s \leftarrow s']$ by r'_1 and r'_2 respectively.

- Base case: $r = t$

$w \models\!\models r$
$\iff$ [By correctness of N as an NFA recognizing $\mathbb{L}(t)$]
If $\epsilon \notin \mathbb{L}(r)$ There exists $\hat{w} \in \mathbb{L}(r)$ and an accepting run $q_0 q_1 \dots q_n$ of N over $\hat{w}$ such that for each $\bar{q_0} \in \{q_I, q_S\}$ and $\bar{q_n} \in Q'_0 \cup \{q_n\}$ there exists a run $w' = s_0 s_1 \dots s_n$ of $D_{N'}$ satisfying w such that

$(1) s_0[state] = \bar{q_0}$
$(2) \forall 1 \le i \le n-1 : s_i[state] = q_i.$
$(3) s_n[state] = \bar{q_n}.$

$\iff$ [By the definitions of idle, start and end]
If $\epsilon \notin \mathbb{L}(r)$ then there exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies
$\{\mathsf{start} \wedge \mathsf{end}\} \cup \{\mathsf{start} \cdot \neg \mathsf{start}^* \cdot \mathsf{end}\}$
otherwise there exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies
$\lambda \cup \{\mathsf{start} \wedge \mathsf{end}\} \cup \{\mathsf{start} \cdot \neg \mathsf{start}^* \cdot \mathsf{end}\}$
$\iff$ There exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies $t''' = (\mathsf{idle} \vee \mathsf{start}) \circ t' \circ (\mathsf{end} \vee \mathsf{idle})$ (if $\epsilon \notin \mathbb{L}(r)$ and $\lambda \cup t'''$otherwise).
$\iff$ There exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies r''.

- Induction step:

(1) $r = r_1 \cdot r_2$

$w \models\!\models r_1 \cdot r_2$
$\iff$ There exist w_1, w_2 s.t. $w = w_1 w_2$, $w_1 \models\!\models r_1$ and $w_2 \models\!\models r_2$
$\iff$ [By the inductive hypothesis]
There exist w_1, w_2 s.t. $w = w_1 w_2$, there exists a run w'_1 of $\mathcal{D}_{N'}$ satisfying w_1 which also satisfies r''_1, and there exists a run w'_2 of $\mathcal{D}_{N'}$ satisfying w_2 which also satisfies r''_2
$\iff$ [By Lemma 23]
There exists w_1, w_2 s.t. $w = w_1 w_2$ and there exists a run $w' = w'_1 w'_2$ of $\mathcal{D}_{N'}$ satisfying $w_1 w_2$ which also satisfies $r''_1 \cdot r''_2$
$\iff$ There exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies r''.

(2) $r = r_1 \cup r_2$

$w \models r_1 \cup r_2$
$\iff w \models r_1$ or $w \models r_2$
$\iff$ [By the inductive hypothesis]
There exists a run w_1 of $\mathcal{D}_{N'}$ satisfying w which also satisfies r_1'' or there exists a run w_1 of $\mathcal{D}_{N'}$ satisfying w which also satisfies r_2''
$\iff$ There exists a run w_1 of $\mathcal{D}_{N'}$ satisfying w which also satisfies r_1'' or r_2''
$\iff$ There exists a run w_1 of $\mathcal{D}_{N'}$ satisfying w which also satisfies $r_1'' \cup r_2''$
$\iff$ There exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies r''.

(3) $r = {r_1}^*$

$w \models {r_1}^*$
$\iff$ Either $w = \epsilon$ or there exist $w_1, w_2, \ldots, w_k$ s.t. $w = w_1 w_2 \ldots w_k$ and for all $1 \leq i \leq k$, $w_i \models r_1$.
$\iff$ [By the inductive hypothesis]
Either $w = \epsilon$ or there exist $w_1, w_2, \ldots, w_k$ s.t. $w = w_1 w_2 \ldots w_k$ and for all $1 \leq i \leq k$ there exists a run w_i' of $\mathcal{D}_{N'}$ satisfying w_i which also satisfies r_1''
$\iff$ [By repetitive applications of Lemma 23]
There exists a run w' of $\mathcal{D}_{N'}$ such that either $|w'| = 0$ or $w' = w_1' w_2' \ldots w_k'$ which satisfies ϵ or $w_1 w_2 \ldots w_k$, respectively, and which also satisfies λ or $r_1'' \cdot r_1'' \cdot \ldots \cdot r_1''$
$\iff$ There exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies ${r_1''}^*$
$\iff$ There exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies r''.

(4) $r = r1 \circ n_2$

$w \models r_1 \circ n_2$
$\iff$ There exist w_1, w_2 and b s.t. $w = w_1 b w_2$, $w_1 b \models r_1$ and $b w_2 \models n_2$
$\iff$ [By the inductive hypothesis]
There exist w_1, w_2 and b s.t. $w = w_1 b w_2$, there exists a run $\sigma_1 s_1$ of $\mathcal{D}_{N'}$ satisfying $w_1 b$ which also satisfies r_1'', and $b w_2 \models n_2$
$\iff$ [By the transition relation of N' and the correctness of $\mathcal{D}_N'$ as its representation]
There exist w_1, w_2 and b s.t. $w = w_1 b w_2$, there exists a run $\sigma_1 s_1$ of $\mathcal{D}_{N'}$ satisfying $w_1 b$ which also satisfies r_1'', and there exists a run $s_2 \sigma_2$ of $\mathcal{D}_{N'}$ satisfying $b w_2$ which also satisfies idle^* and n_2
$\iff$ [By Lemma 24]
There exist w_1, w_2 and b s.t. $w = w_1 b w_2$ and there exists a run $w' = \sigma_1 s_1 \sigma_2$ of $\mathcal{D}_{N'}$ satisfying $w_1 b w_2$ which also satisfies $r_1'' \circ n_2$
$\iff$ There exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies r''.

(5) $r = n1 \circ r_2$
Symmetric to the above case.

(6) $r = r1 \cap n_2$

$w \models r_1 \cap n_2$
$\iff w \models r_1$ and $w \models n_2$
$\iff$ [By the inductive hypothesis]
There exists a run w_1 of $\mathcal{D}_{N'}$ satisfying w which also satisfies r_1'' and $w \models n_2$
$\iff$ There exists a run w_1 of $\mathcal{D}_{N'}$ satisfying w which also satisfies r_1'' and n_2
$\iff$ There exists a run w_1 of $\mathcal{D}_{N'}$ satisfying w which also satisfies $r_1'' \cap n_2$
$\iff$ There exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies r''.

(7) $r = r1 \cap n_2$
Symmetric to the above case. □

Lemma 23. Let σ_1', σ_2' be runs of $D_{N'}$ satisfying $(\mathsf{idle} \vee \mathsf{start}) \circ \mathit{true}^* \circ (\mathsf{end} \vee \mathsf{idle})$ as well as w_1 and w_2, respectively. Then $\sigma_1' \sigma_2'$ is a run of $\mathcal{D}_{N'}$ satisfying $(\mathsf{idle} \vee \mathsf{start}) \circ \mathit{true}^* \circ (\mathsf{end} \vee \mathsf{idle})$ as well as $w_1 w_2$.

Proof. A run of $D_{N'}$ satisfying $(\mathsf{idle} \vee \mathsf{start}) \circ \mathit{true}^* \circ (\mathsf{end} \vee \mathsf{idle})$ corresponds to a run of N' starting with state q_I or state $q_0 \in Q_0$ and ending in state $q \in \mathit{pred}(A)$ or in state q_I. Denote by σ_1, σ_2 the runs of N' corresponding to σ_1', σ_2' respectively. σ_2 starts in $q_0 \in Q_0'$. Let b_0 be the first letter of w_2, then there is a transition from $q \in \mathit{pred}(A) \cup \{q_i\}$ to q_0 via b_0. Thus the concatenation $\sigma_1 \sigma_2$ is possible. The run $\sigma' \sigma_2'$ starts with a state s_0 such that $s_0[\mathit{state}] = q_0 \in Q_0$ or $s_0[\mathit{state}] = q_I$ and ends in a state s_n such that $s_n[\mathit{state}] \in \mathit{pred}(A) \cup \{q_I\}$. It thus satisfies $(\mathsf{idle} \vee \mathsf{start}) \circ \mathit{true}^* \circ (\mathsf{end} \vee \mathsf{idle})$. Clearly if σ_1' satisfies w_1 and σ_2 satisfies w_2, the concatenated run $\sigma_1' \sigma_2'$ satisfies the concatenated word $w_1 w_2$. □

Lemma 24.
- Let $\sigma'_1 s_1$ be a run of $D_{N'}$ satisfying (idle $\vee$ start) $\circ$ *true** $\circ$ (end $\vee$ idle) as well as $w_1 b$. Let $s_2\sigma'_2$ be a run of $D_{N'}$ satisfying (idle$*$) as well as bw_2. Then $\sigma'_1 s_1 \sigma'_2$ is a run of $\mathcal{D}_{N'}$ satisfying (idle $\vee$ start) $\circ$ *true** $\circ$ (end $\vee$ idle) as well as $w_1 b w_2$.
- Let $\sigma'_1 s_1$ be a run of $D_{N'}$ satisfying (idle$*$) as well as $w_1 b$. Let $s_2\sigma'_2$ be a run of $D_{N'}$ satisfying (idle $\vee$ start) $\circ$ *true** $\circ$ (end $\vee$ idle) as well as bw_2. Then $\sigma'_1 s_2 \sigma'_2$ is a run of $\mathcal{D}_{N'}$ satisfying (idle $\vee$ start) $\circ$ *true** $\circ$ (end $\vee$ idle) as well as $w_1 b w_2$.

Proof. We show only the first case, the second case is symmetric. The run $\sigma'_1 s_1$ of $D_{N'}$ corresponds to a run σq_1 of N' where $s_1[state] = q_1$ and $\sigma_1 q_1$ starts with a state $q_0 \in Q_0 \cup \{q_I\}$ and ends in state $q \in pred(A) \cup \{q_I\}$. The run $s_2\sigma'_2$ of $D_{N'}$ corresponds to a run $q_I\sigma_2$ of N' of the form q_I^*. Since from every $q \in pred(A) \cup \{q_I\}$ there are transitions to q_I, we can concatenate the second run, after chopping its first state to the end of the first run. The resulting run $\sigma_1 q_1 \sigma_2$, thus, starts with a state $q_0 \in Q_0 \cup \{q_I\}$ and ends in a state q_I (or in state $q \in pred(A)$, if $|\sigma_2| = 0$). Therefore $\sigma'_1 s_1 \sigma'_2$ satisfies (idle $\vee$ start) $\circ$ *true** $\circ$ (end $\vee$ idle). Clearly the concatenated run $\sigma'_1 s_1 \sigma'_2$ satisfies the concatenated word $w_1 b w_2$. □

A.3. Proof of Proposition 25

Lemma 31. *Let r be a* SERE*, and $t = r_1 \cap r_2$ a sub-*SERE *of r. Let N'_1 and N'_2 be the cruising* NFAs *for r_1 and r_2 respectively. Let $\mathcal{D}_{N'_1}$ and $\mathcal{D}_{N'_1}$ be their corresponding* DTSs*.*
Let $idle_1$ and $idle_2$ denote, respectively, that $D_{N'_1}$ and $D_{N'_2}$ are in a state s such that $s[state] = q_I$. Let $idle$ denote the conjunction $idle_1 \wedge idle_2$. Let t' denote the SERE *placeholder*$(r_1 \cap r_2, \text{start}, \text{middle}, \text{end})$*. Let r''' denote the* SERE $(idle \vee \text{start}) \circ r[t \leftarrow t'] \circ (\text{end} \vee idle)$ *and let r'' denote the* SERE *r''' if $\epsilon \notin \mathbb{L}(r)$ and the* SERE *$\lambda \cup r'''$ otherwise. Then*

$$w \models r \quad \Updownarrow \quad \text{there exists a run of } \mathcal{D}_{N'_1} ||| \mathcal{D}_{N'_2} \text{ satisfying } w \text{ which satisfies also } r''.$$

Proof. We show only the base case ($r = r_1 \cap r_2$). The rest of the proof proceeds similarly to the proof of Lemma 22.

$w \models r_1 \cap r_2$

$\iff$ $w \models r_1$ and $w \models r_2$

$\iff$ [By Lemma 22]

If $\epsilon \notin \mathbb{L}(r_1)$ then there exists a run w' of $\mathcal{D}_{N'_1}$ satisfying w which also satisfies

$$\{\text{start}_1 \wedge \text{end}_1\} \cup \{\text{start}_1 \cdot \neg \text{start}_1{}^* \cdot \text{end}_1\}.$$

otherwise there exists a run w' of $\mathcal{D}_{N'_1}$ satisfying w which also satisfies

$$\lambda \cup \{\text{start}_1 \wedge \text{end}\}_1 \cup \{\text{start}_1 \cdot \neg \text{start}_1{}^* \cdot \text{end}_1\}$$

And if $\epsilon \notin \mathbb{L}(r_2)$ then there exists a run w' of $\mathcal{D}_{N'_2}$ satisfying w which also satisfies

$$\{\text{start}_2 \wedge \text{end}_2\} \cup \{\text{start}_2 \cdot \neg \text{start}_2{}^* \cdot \text{end}_2\}$$

otherwise there exists a run w' of $\mathcal{D}_{N'_2}$ satisfying w which also satisfies

$$\lambda \cup \{\text{start}_2 \wedge \text{end}_2\} \cup \{\text{start}_2 \cdot \neg \text{start}_2{}^* \cdot \text{end}_2\}$$

$\iff$ If $\epsilon \notin \mathbb{L}(r_1 \cap r_2)$ then there exists a run w' of $\mathcal{D}_{N'_1} ||| \mathcal{D}_{N'_2}$ satisfying w which also satisfies

$$\{(\text{start}_1 \wedge \text{start}_2) \wedge (\text{end}_1 \wedge \text{end}_2)\} \cup \{\text{start}_1 \wedge \text{start}_2) \cdot (\neg \text{start}_1 \wedge \neg \text{start}_2)^* \cdot (\text{end}_1 \wedge \text{end}_2)\}$$

otherwise there exists a run w' of $\mathcal{D}_{N'_1} ||| \mathcal{D}_{N'_2}$ satisfying w which also satisfies

$$\lambda \cup \{(\text{start}_1 \wedge \text{start}_2) \wedge (\text{end}_1 \wedge \text{end}_2)\} \cup \{\text{start}_1 \wedge \text{start}_2) \cdot (\neg \text{start}_1 \wedge \neg \text{start}_2)^* \cdot (\text{end}_1 \wedge \text{end}_2)\}$$

$\iff$ If $\epsilon \notin \mathbb{L}(r_1 \cap r_2)$ then there exists a run w' of $\mathcal{D}_{N'_1} ||| \mathcal{D}_{N'_2}$ satisfying w which also satisfies

$$\{\text{start} \wedge \text{end}\} \cup \{\text{start} \cdot \text{middle}^* \cdot \text{end}\}$$

otherwise there exists a run w' of $\mathcal{D}_{N'_1} ||| \mathcal{D}_{N'_2}$ satisfying w which also satisfies

$$\lambda \cup \{\text{start} \wedge \text{end}\} \cup \{\text{start} \cdot \text{middle}^* \cdot \text{end}\}$$

$\iff$ There exists a run of $\mathcal{D}_{N'_1} ||| \mathcal{D}_{N'_2}$ satisfying w which satisfies also r''. □

Proposition 25.
Let $\mathcal{D}_M$ be a DTS, r a SERE and $r_1 \cap r_2$ a sub-SERE of r. Let N'_1 and N'_2 be the cruising NFAs , for r_1 and r_2 respectively. Let $\mathcal{D}_{N'_1}$ and $\mathcal{D}_{N'_1}$ be their corresponding DTSs. Then

$$\mathcal{D}_M \models \text{not } r!$$
$$\Updownarrow$$
$$\mathcal{D}_M \,|||\, \mathcal{D}_{N'_1} \,|||\, \mathcal{D}_{N'_2} \models \text{not } r[r_1 \cap r_2 \leftarrow \mathit{placeholder}(r_1 \cap r_2, \text{start}, \text{middle}, \text{end})]!$$

Proof. The proof is similar to the proof of Theorem 18, this time making use of Lemma 31. □

A.4. Proof of Proposition 29

For completeness of the proof of Proposition 29 we give here the semantics of CTL formulas and the semantics of not r! over a model and a state in the model (rather than over a path).

Definition 32 (*Semantics of* CTL *Formulas*). The semantics of CTL formulas are defined with respect to a model (DTS) and a state in the model. Let $\mathcal{D}_M = \langle V_M, \Theta_M, \rho_M, \mathcal{A}_M \rangle$ be a DTS. The notation $\mathcal{D}_M, s \models \varphi$ means that formula φ holds in state s of model $\mathcal{D}_M$. The notation $\mathcal{D}_M \models \varphi$ is equivalent to $\mathcal{D}_M, s \models \varphi$ for all s such that $s \models \Theta$. In other words, φ is valid for every initial state of $\mathcal{D}_M$. The semantics of a CTL formula over D_M are defined as follows, where b denotes a boolean expression and φ, φ_1, and φ_2 denote CTL formulas.

- $\mathcal{D}_M, s \models b \iff s \models b$
- $\mathcal{D}_M, s \models \neg\varphi \iff \mathcal{D}_M, s \not\models \varphi$
- $\mathcal{D}_M, s \models \varphi_1 \wedge \varphi_2 \iff \mathcal{D}_M, s \models \varphi_1$ and $\mathcal{D}_M, s \models \varphi_2$
- $\mathcal{D}_M, s \models EX\,\varphi \iff \exists$ run σ of $\mathcal{D}_M$ s.t. $|\sigma| > 1, \sigma^0 = s$, and $\mathcal{D}_M, \sigma^1 \models \varphi$
- $\mathcal{D}_M, s \models E[\varphi_1\, U\, \varphi_2] \iff \exists$ run σ of $\mathcal{D}_M$ s.t.$\sigma_0 = s$ and $\exists k < |\sigma|$ s.t. $\mathcal{D}_M, \sigma^k \models \varphi_2$ and $\forall j$ s.t. $j < k$: $\mathcal{D}_M, \sigma^j \models \varphi_1$
- $\mathcal{D}_M, s \models EG\,\varphi \iff \exists$ run σ of $\mathcal{D}_M$ s.t. $\sigma^0 = s$ and $\forall j$ s.t. $0 \le j < |\sigma|$: $\mathcal{D}_M, \sigma_j \models \varphi$.

Proposition 29. Let $\mathcal{D}_M$ be a discrete system and r a SERE with no starred sub-SEREs, such that $\epsilon \not\models r$. Then, $\mathcal{D}_M \models \text{not } r! \iff \mathcal{D}_M \models \neg T(r)$.

Proof. First we note that the translation procedure (Definition 28) covers all SEREs r such that $\epsilon \not\models r$ and r does not contain any starred sub-SEREs. Second, we note that Definition 30 is legitimate, since although SEREs are defined over non-empty words, the definition $\mathcal{D}_M \models \text{not } r!$ relies on the fact that runs are by definition non-empty and further assumes $\epsilon \not\models r$. From the same reason we can first show that for every $s \in \Sigma_V$,

$$\mathcal{D}_M, s \models \text{not } r! \iff \mathcal{D}_M, s \models \neg T(r).$$

This is shown by induction on the structure of r, as follows. Let b be a boolean expression and let r_1, r_2 be SEREs.

- Base case.

1. $\mathcal{D}_M, s \models \text{not } \{b\}!$
$\iff \mathcal{D}_M, s \models \neg b$
$\iff \mathcal{D}_M, s \models \neg T(b)$.

- Induction step.

2. $\mathcal{D}_M, s \models \text{not } \{r \cdot b^*\}!$
$\iff \mathcal{D}_M, s \models \text{not } r!$
$\iff$ [by the induction hypothesis] $\mathcal{D}_M, s \models \neg T(r)$.

3. $\mathcal{D}_M, s \models \text{not } \{r_1 \cup r_2\}!$
$\iff \mathcal{D}_M, s \models \text{not } \{r_1\}! \wedge \text{not } \{r_2\}!$
$\iff$ [by the induction hypothesis] $\mathcal{D}_M, s \models \neg T(r_1) \wedge \neg T(r_2)$
$\iff \mathcal{D}_M, s \models \neg T(r_1 \cup r_2)$.

4. $\mathcal{D}_M, s \models \text{not } \{b \cdot r_1\}!$
$\iff$ for every finite run $\sigma = ss'\sigma'$, $s \models \neg b$ or $\mathcal{D}_M, s' \models \text{not } \{r_1\}!$
$\iff$ [by the induction hypothesis]
$s \models \neg b$ or for every $\sigma = ss'\sigma'$, $\mathcal{D}_M, s' \models \neg T(r_1)$

$\iff \mathcal{D}_M, s \models \neg b \vee \mathsf{AX}\neg T(r_1)$
$\iff \mathcal{D}_M, s \models \neg T(b{\cdot}r_1)$.

5. $\mathcal{D}_M, s \not\models \text{not } \{b^*{\cdot}r_1\}$!
$\iff$ there exists a run σ s.t. either $\sigma \not\models r_1$ or $\sigma = s^0 s^1 \ldots s^n\sigma'$ (where $s^0 = s$) and $\forall 0 \leq j < n : \sigma^j \models b$ and $\sigma^n\sigma' \not\models r_1$
$\iff \mathcal{D}_M, s \not\models \text{not } r!$ or there exists a run $\sigma = s^0 s^1 \ldots s^n\sigma'$ (where $s^0 = s$) and $\forall 0 \leq j < n : \sigma^j \models b$ and $\mathcal{D}_M, \sigma^n \not\models \text{not } \{r_1\}$!
$\iff$ [by the induction hypothesis]
$\mathcal{D}_M, s \models T(r)$ or there exists a run $\sigma = s^0 s^1 \ldots s^n\sigma'$ (where $s^0 = s$) and $\forall 0 \leq j < n : \sigma^j \models b$ and $\mathcal{D}_M, \sigma^n \models T(r_1)$
$\iff \mathcal{D}_M, s \models T(r_1) \vee E[b\ U\ T(r_1)]$
$\iff \mathcal{D}_M, s \models E\,[b\ U\ T(r_1)]$
$\iff \mathcal{D}_M, s \models T(b^*{\cdot}r_1)$.

6. $\mathcal{D}_M, s \models \text{not } \{r_1 \cup r_2{\cdot}r\}$!
$\iff \mathcal{D}_M, s \models \text{not } \{r_1{\cdot}r\}! \wedge \text{not } \{r_2{\cdot}r\}!$
$\iff$ [by the induction hypothesis]
$\mathcal{D}_M, s \models \neg T(r_1{\cdot}r) \wedge \neg T(r_2{\cdot}r)$
$\iff \mathcal{D}_M, s \models \neg T(r_1 \cup r_2{\cdot}r)$.

In particular for every initial state s, $\mathcal{D}_M, s \models \text{not } r! \iff \mathcal{D}_M, s \models \neg T(r)$. Therefore $\mathcal{D}_M \models \text{not } r! \iff \mathcal{D}_M \models \neg T(r)$. □

References

[1] Y. Abarbanel, I. Beer, L. Gluhovsky, S. Keidar, Y. Wolfsthal, FoCs — automatic generation of simulation checkers from formal specifications, in: Proc. 12th International Conference on Computer Aided Verification, CAV, in: LNCS, vol. 1855, Springer-Verlag, 2000.
[2] V.M. Antimirov, P.D. Mosses, Rewriting extended regular expressions, Theoretical Computer Science 143 (1) (1995) 51–72.
[3] I. Beer, S. Ben-David, C. Eisner, D. Geist, L. Gluhovsky, T. Heyman, A. Landver, P. Paanah, Y. Rodeh, G. Ronin, Y. Wolfsthal, RuleBase: Model checking at IBM, in: Proc. 9th International Conference on Computer Aided Verification, CAV, in: LNCS, vol. 1254, Springer-Verlag, 1997, pp. 480–483.
[4] I. Beer, S. Ben-David, A. Landver, On-the-fly model checking of RCTL formulas, in: Proc. 10th International Conference on Computer Aided Verification, CAV'98, in: LNCS, vol. 1427, Springer-Verlag, 1998, pp. 184–194.
[5] S. Ben-David, D. Fisman, S. Ruah, The Safety Simple Subset, Haifa Verification Conference, HVC, in: LNCS, vol. 3875, Springer-Verlag, 2005, pp. 14–29.
[6] S. Ben-David, D. Fisman, S. Ruah, Automata construction for regular expressions in model checking, June 2004. IBM research report H-0229.
[7] R. Bloem, K. Ravi, F. Somenzi, Efficient decision procedures for model checking of linear time logic properties, in: Computer Aided Verification, 1999, pp. 222–235.
[8] D. Drusinsky, D. Harel, On the power of bounded concurrency I: Finite automata, Journal of the ACM 41 (3) (1994) 517–539.
[9] E. Emerson, Temporal and modal logics, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, vol. B, Elsevier, 1990, pp. 995–1072.
[10] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, in: Addison-Wesley Series in Computer Science, Addison-Wesley, 1979.
[11] IEEE Standard for Property Specification Language (PSL). IEEE Std 1850™-2005.
[12] Y. Kesten, A. Pnueli, L. Raviv, Algorithmic verification of linear temporal logic specifications, in: K. Larsen, S. Skyum, G. Winskel (Eds.), Proc. 25th Int. Colloq. Aut. Lang. Prog., in: Lect. Notes in Comp. Sci., vol. 1443, Springer-Verlag, 1998, pp. 1–16.
[13] O. Kupferman, Augmenting branching temporal logics with existential quantification over atomic propositions, Journal of Logic and Computation 7 (1997) 1–14.
[14] M. Maidl, The common fragment of CTL and LTL, in: IEEE Symposium on Foundations of Computer Science, 2000, pp. 643–652.
[15] K. McMillan, Symbolic model checking, 1993.
[16] A.C. Patthak, I. Bhattacharya, A. Dasgupta, P. Dasgupta, P.P. Chakrabarti, Quantified Computation Tree Logic 82 (3) (2002) 123–129.
[17] A. Pnueli, A temporal logic of concurrent programs, Theoretical Computer Science 13 (1981) 45–60.
[18] P. Wolper, Temporal logic can be more expressive, in: 22nd Annual Symposium on Foundation of Computer Science, October 1981.